EECE7205 Fundamentals of Computer Engineering

Project 1 Report

Name: Nikita Vinod Mandal

NUID: 002826995

- **Problem Description:**

You are given an input array $A[1,\ldots,N]$. A grouping of the array A is described by an array $G[1,\ldots,M]$, where the array A is partitioned into M groups, the 1st group consists of the first $G[1]$ elements of array A, the 2nd group consists of the next $G[2]$ elements, and so forth. Define array $B[1,\ldots,M]$ such that $B[j]$ is the summation of the elements in the j-th group of array A. Use a dynamic programming algorithm to find a grouping of array A with M groups such that we maximize the minimum element of array B.

Max-min-grouping(A, N, M) {

return $G[1,\ldots,M]$ }

- **Pseudo Code:**

//Define constant values for the maximum possible sizes of the input arrays and dynamic programming tables. We do this to indicate the maximum limit of the problem

```
const int MAX_N = 1000;

const int MAX_M = 1000;
```

//Initialize the dynamic programming table and the array to store group sizes. This array is used to store the dynamic programming table for solving the problem.

```
int darr[MAX_N + 1][MAX_M + 1];

int G[MAX_M];


// Dynamic programming for finding optimal solutions

function MaxMinGrouping(A[], N, M):

    // Initialize the first row for prefix sum

    for i from 1 to N:
        darr[i][1] = darr[i - 1][1] + A[i - 1]

    // Perform dynamic programming to find optimal solutions
    for j from 2 to M:
        for i from j to N:
            for k from j - 1 to i - 1:
                temp = min(darr[k][j - 1], darr[i][1] - darr[k][1])
                darr[i][j] = max(darr[i][j], temp)

    // Backtrack to find group sizes
    i = N
    j = M
    while j > 0:
        for k from 0 to i:
            if darr[i][j] equals darr[k][j - 1] or darr[i][j] equals darr[i][1] - darr[k][1]:
                G[j - 1] = i - k
                i = k
                break
        j--

    // Output the maximum minimum value of B
    output "The Maximum minimum value of B is : " + darr[N][M]

    // Return the result as an array. This gives the output array of optimal grouping.
    result = [G[0], G[1], ..., G[M-1]]
    return result

// Main function
function main():
    input
```

```
output "Enter the number of elements in array A: "
input A[N]
output "Enter the number of groups (M) needed: "
input M

// Call the MaxMinGrouping function to find the optimal grouping
G = MaxMinGrouping(A, N, M)

// Output the optimal grouping
output "The Optimal grouping is: "
for group in G:
    output group + " "
output "\nThe Elements in each group are:"

nz = 0
for group in G:
    output "Group :"
    for i from 0 to group - 1:
        output A[nz++] + " "
return 0
```

- **Analysis of the running time asymptotically:**

To analyze the running time of the provided code asymptotically, we will break it down in sub parts.

1. **Complexity for Dynamic Programming Part:**

The dynamic programming part of the code consists of three nested loops. The outer loop runs for M iterations, the middle loop runs for N iterations, and the innermost loop runs for at most N iterations. The time complexity of the dynamic programming part is approximately **O(M * N^2).**

2. **Complexity for Backtracking Part:**

The backtracking loop runs for at most M iterations, and the inner loop runs for at most N iterations. The time complexity of the backtracking part is approximately **O(M * N).**

3. **Complexity for I/O Operations:**

Input and output operations are typically considered to be **O(1)** as they do not depend on the size of the input but rather on the number of elements read/written.

4. **Complexity of the overall code:**

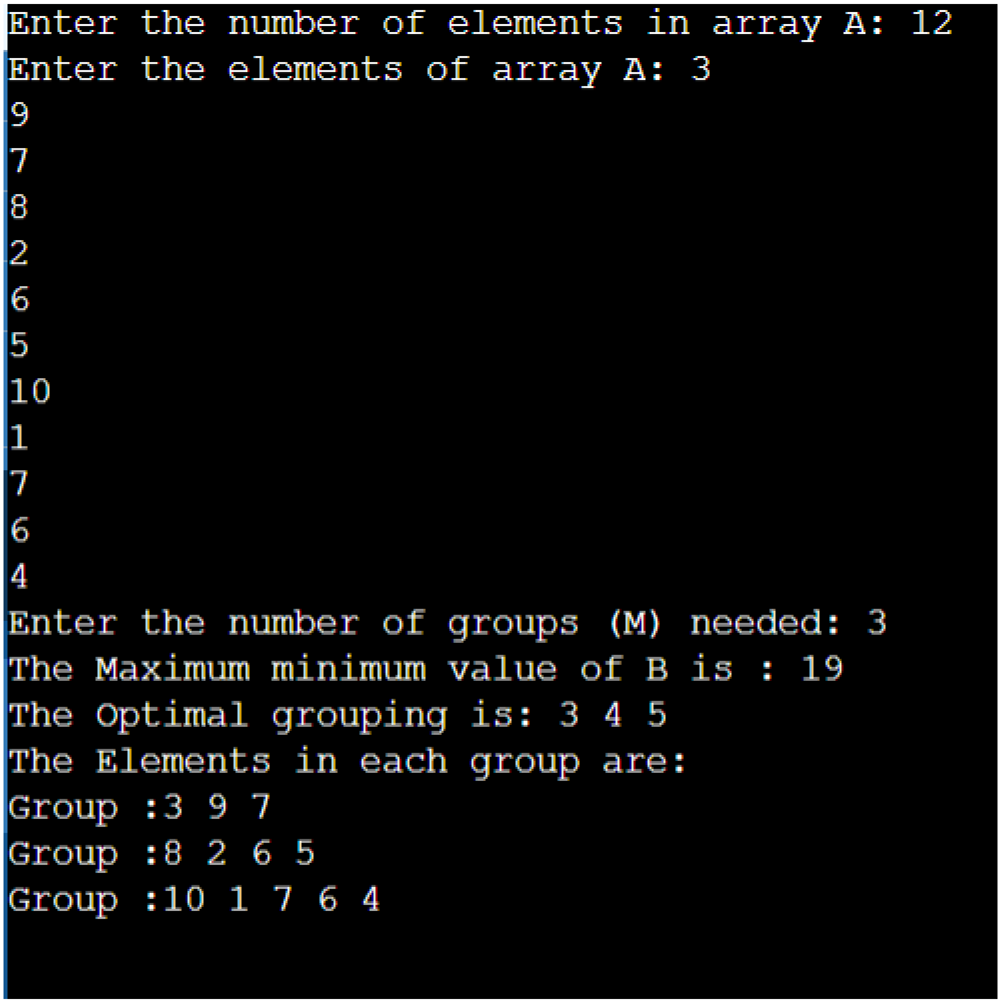
Overall, the most significant and dominant factor that determines the time complexity is the dynamic programming part with a time complexity of **O(M * N^2).**

Here, N is the number of elements in array A, and M is the number of groups needed. This complexity indicates that the code's execution time increases quadratically with the size of the input array and linearly with the number of groups.

- **Grouping results of several input examples:**

1. **Input 1:** A={3,9,7,8,2,6,5,10,1,7,6,4} and M=3

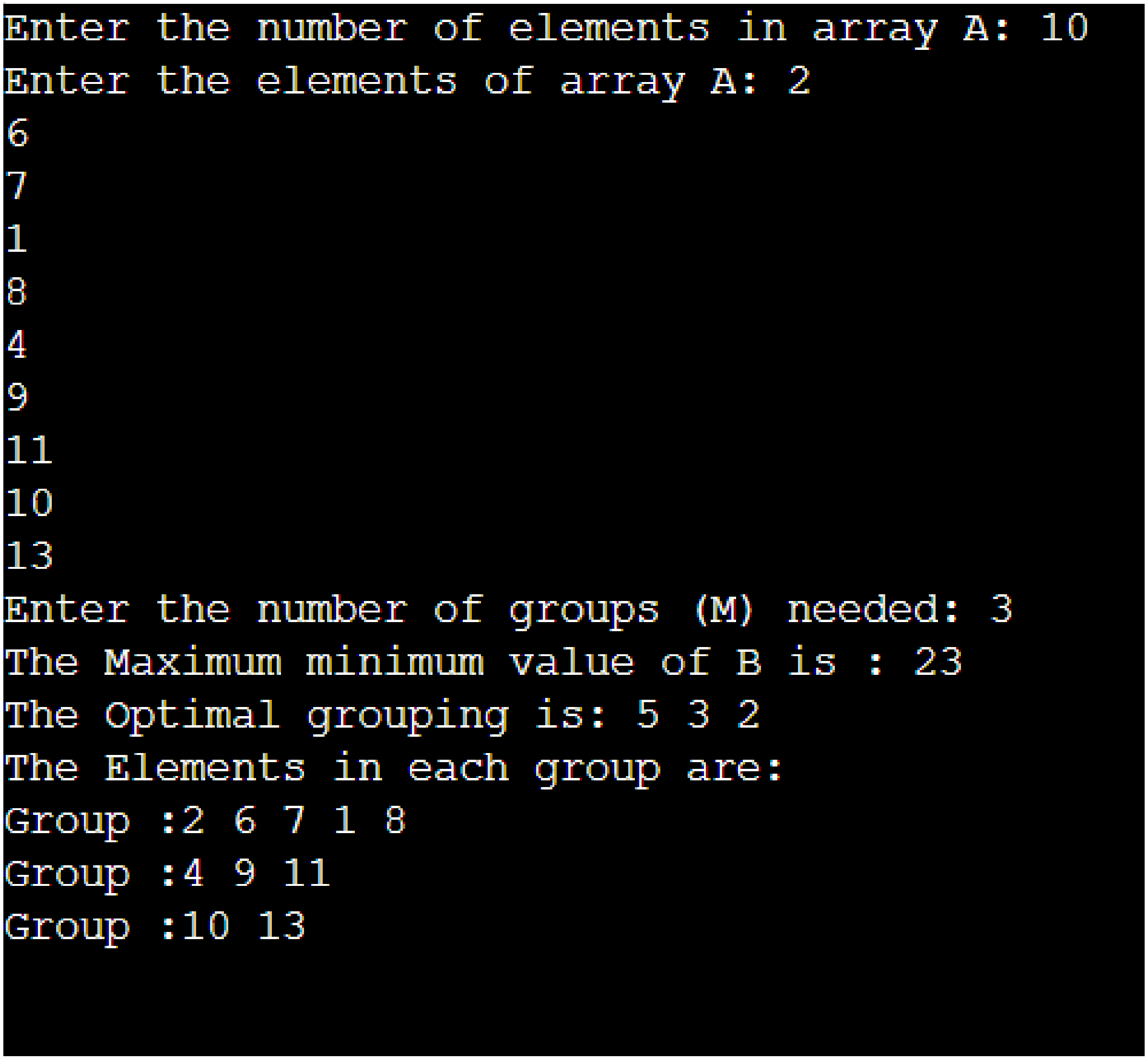
Output:



```
Enter the number of elements in array A: 12
Enter the elements of array A: 3
9
7
8
2
6
5
10
1
7
6
4
Enter the number of groups (M) needed: 3
The Maximum minimum value of B is : 19
The Optimal grouping is: 3 4 5
The Elements in each group are:
Group :3 9 7
Group :8 2 6 5
Group :10 1 7 6 4
```

2. **Input 2:** A={2,6,7,1,8,4,9,11,10,13} and M=3

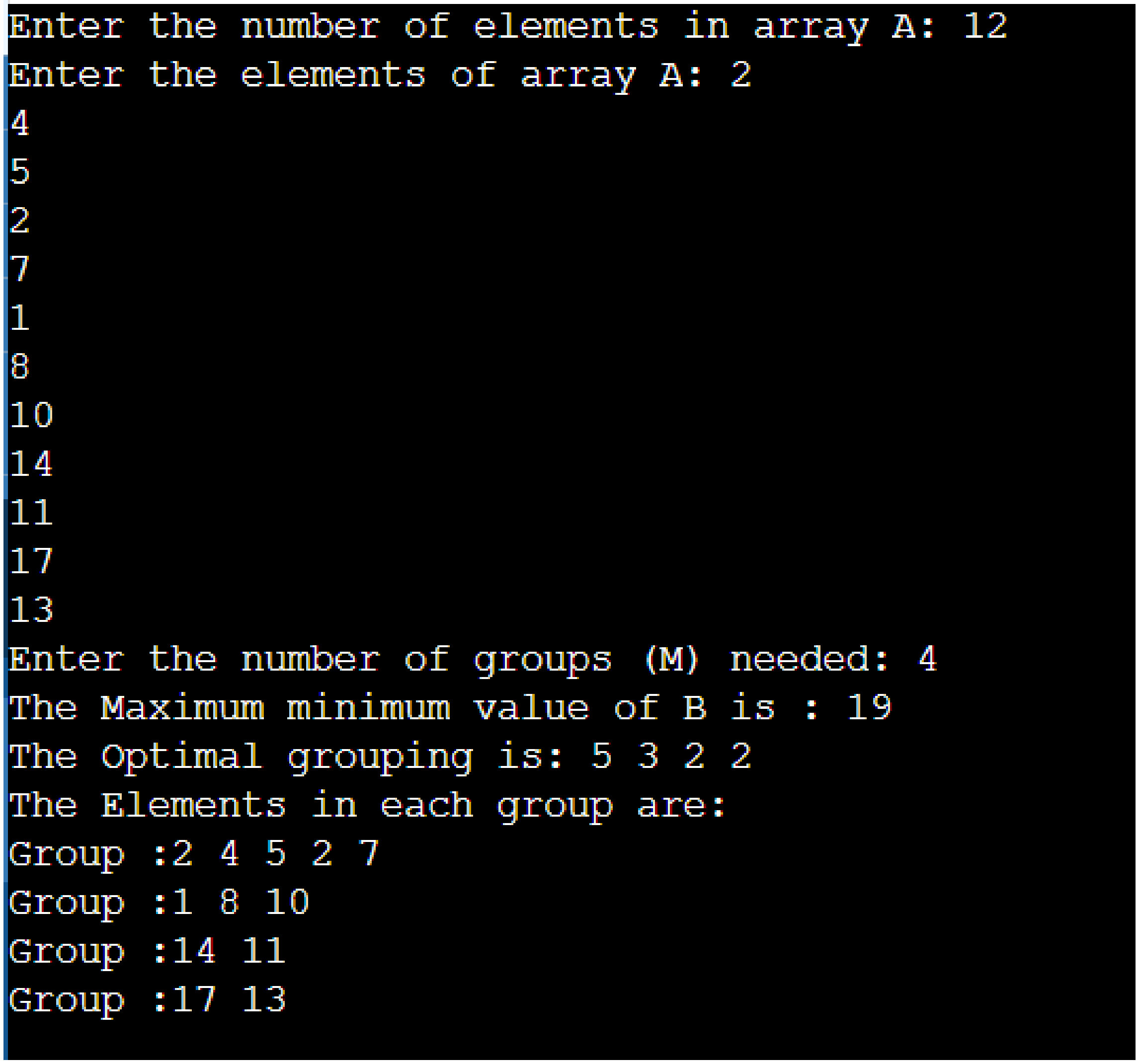
Output:



```
Enter the number of elements in array A: 10
Enter the elements of array A: 2
6
7
1
8
4
9
11
10
13
Enter the number of groups (M) needed: 3
The Maximum minimum value of B is : 23
The Optimal grouping is: 5 3 2
The Elements in each group are:
Group :2 6 7 1 8
Group :4 9 11
Group :10 13
```

3. **Input 3:** A={2,4,5,2,7,1,8,10,14,11,17,13} and M=4

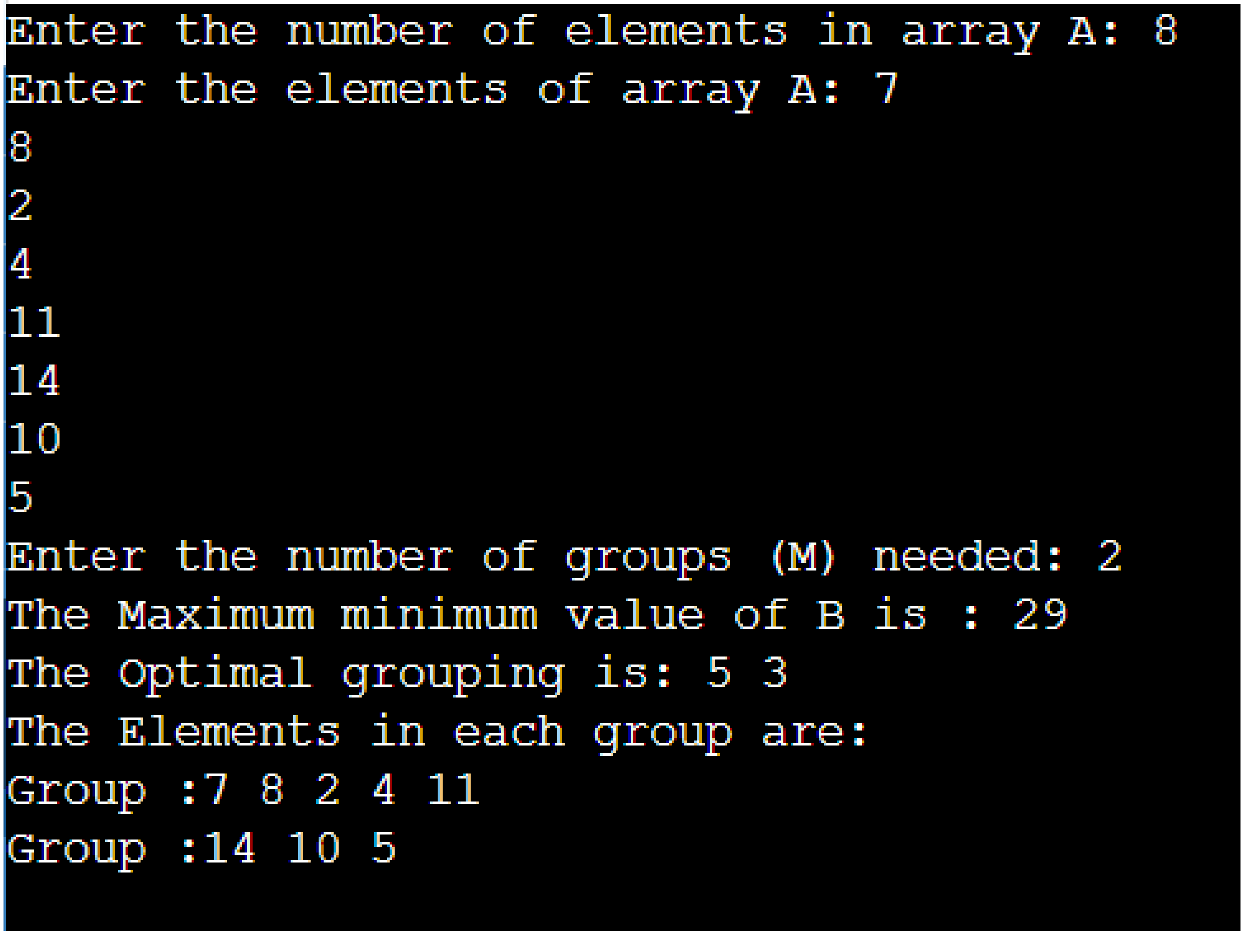
Output:



```
Enter the number of elements in array A: 12
Enter the elements of array A: 2
4
5
2
7
1
8
10
14
11
17
13
Enter the number of groups (M) needed: 4
The Maximum minimum value of B is : 19
The Optimal grouping is: 5 3 2 2
The Elements in each group are:
Group :2 4 5 2 7
Group :1 8 10
Group :14 11
Group :17 13
```

4. **Input 4:** A={7,8,2,4,11,14,10,5} and M=2

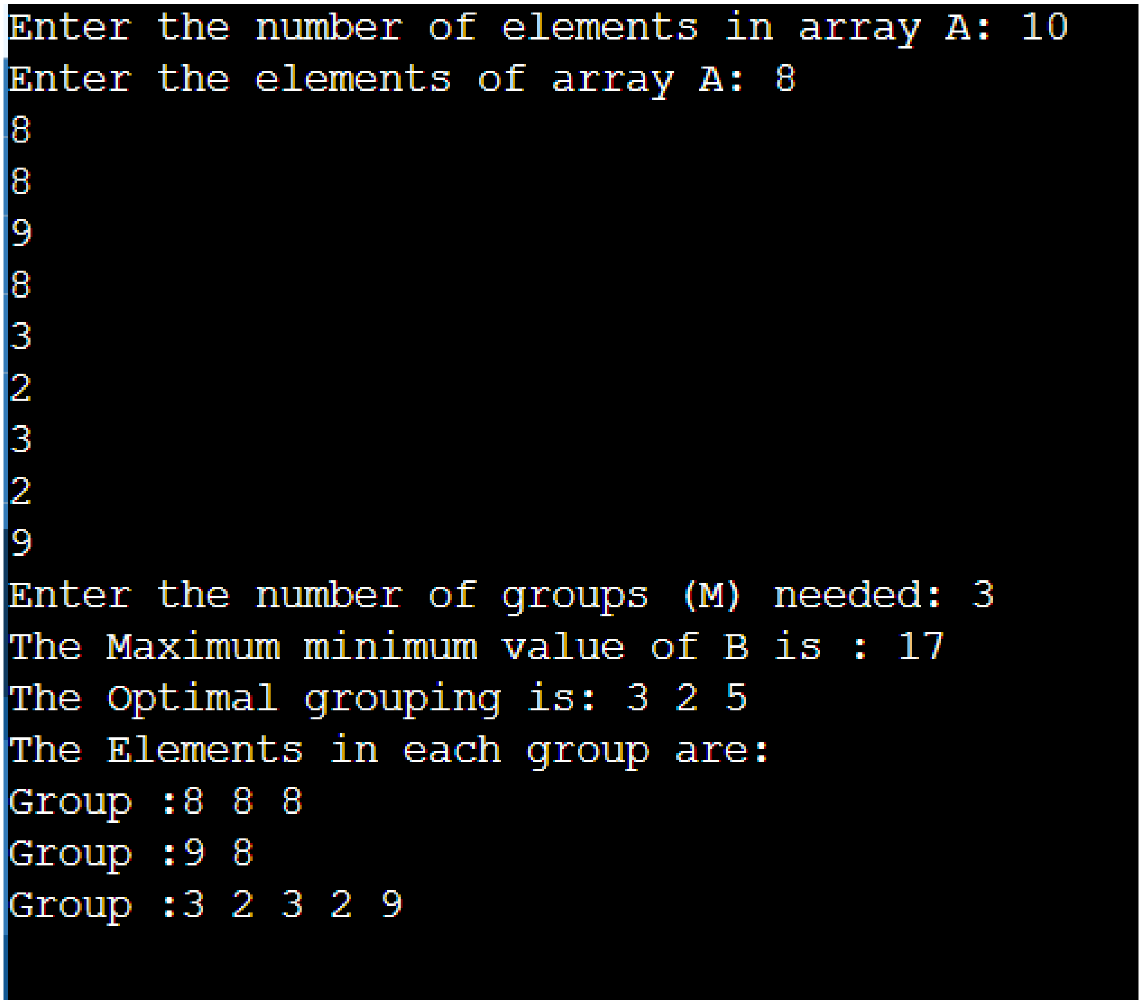
Output:



```
Enter the number of elements in array A: 8
Enter the elements of array A: 7
8
2
4
11
14
10
5
Enter the number of groups (M) needed: 2
The Maximum minimum value of B is : 29
The Optimal grouping is: 5 3
The Elements in each group are:
Group :7 8 2 4 11
Group :14 10 5
```

5. **Input 5:** A={8,8,8,9,8,3,2,3,2,9} and M=3

Output:



```
Enter the number of elements in array A: 10
Enter the elements of array A: 8
8
8
9
8
3
2
3
2
9
Enter the number of groups (M) needed: 3
The Maximum minimum value of B is : 17
The Optimal grouping is: 3 2 5
The Elements in each group are:
Group :8 8 8
Group :9 8
Group :3 2 3 2 9
```

- **Source Code:**

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

//define constant values for the maximum possible sizes of the input arrays and dynamic programming tables
const int MAX_N = 1000;
const int MAX_M = 1000;

//initialize the first row and column for the dynamic programming
int darr[MAX_N + 1][MAX_M + 1];
int G[MAX_M];

//dynamic programming for optimal solutions
vector<int> MaxMinGrouping(int A[], int N, int M)
{
    for (int i = 1; i <= N; ++i)
    {
        darr[i][1] = darr[i - 1][1] + A[i - 1];
    }

    for (int j = 2; j <= M; ++j)
    {
        for (int i = j; i <= N; ++i)
        {
            for (int k = j - 1; k < i; ++k)
            {
                int temp = min(darr[k][j - 1], darr[i][1] - darr[k][1]);
                darr[i][j] = max(darr[i][j], temp);
            }
        }
    }

    int j = M, i = N;
    while (j > 0)
    {
        for (int k = 0; k < i; ++k)
        {
            if (darr[i][j] == darr[k][j - 1] || darr[i][j] == darr[i][1] - darr[k][1])
            {
                G[j - 1] = i - k;
                i = k;
                break;
            }
        }
        j--;
    }
    cout << "The Maximum minimum value of B is : " << darr[N][M] << endl;

    vector<int> result(G, G + M);
    return result;
}

int main()
{
    int N;
    cout << "Enter the number of elements in array A: ";
    cin >> N;
    int A[MAX_N];
    cout << "Enter the elements of array A: ";
    for (int i = 0; i < N; ++i)
    {
        cin >> A[i];
    }

    int M;
    cout << "Enter the number of groups (M) needed: ";
    cin >> M;

    vector<int> G = MaxMinGrouping(A, N, M);

    cout << "The Optimal grouping is: ";
    for (int group : G)
    {
        cout << group << " ";
    }
    cout << "\nThe Elements in each group are:" << endl;

//ensure all the groups are non zero
    int nz = 0;
    for (int group : G)
    {
        cout << "Group :" ;
        for (int i = 0; i < group; ++i)
        {
            cout << A[nz++] << " ";
        }
        cout << endl;
    }
    return 0;
}
```